INF 385T · FALL 2026

TEXAS
The University of Texas at Austin

INF385T: Special Topics in Information Science: Language Models and Agentic Workflows

Week 5: Single Agent Loops and Adaptive Control

Dr. Abhijit Mishra

Announcement

- Group Proposal Due Today
- Assignment 3 posted:
  - **Assignment 3:** The Bounded Agent Rescue Mission
  - Based on this week (NOT last week's lecture + tutorial)

Week 4: Recap

- Deterministic Workflows vs Autonomous agents
- Prompt chaining and conditional routing
- Parallelization and aggregation
- Evaluator–optimizer patterns
- State machines and task graphs
- Deterministic control around nondeterministic models.

Recap : Deterministic Workflows

- A workflow is a controlled sequence or graph of stages that specifies how a task moves from input to completion.
- Typically has:
  - Ordered or graph-structured stages
  - Defined inputs and outputs
  - Explicit transitions between stages
  - Known control logic
  - Clear terminal conditions
  - Observable execution state

Recap: Design the workflow before prompting



Recap: Core Workflow Patterns

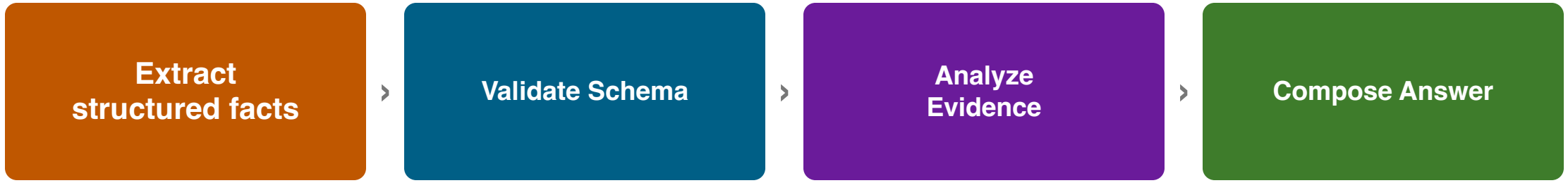
- **Typed stage boundaries** (Named input and output fields, Required and optional values, Closed status values and explicit errors, evidence **provenance)**
- Preconditions, postconditions, invariants, and state
- Observable transitions
- Prompt Chaining
- Conditional Routing

Prompt chaining

- Split one compound model request into sequential bounded stages
- Output from one stage becomes validated input to the next
- Appropriate when later work depends on a preceding artifact

Recap: Prompt chaining

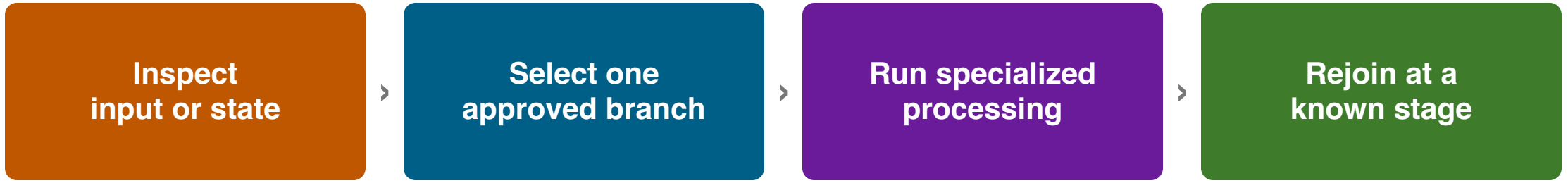
Split one compound model request into sequential bounded stages



Each stage has one responsibility. A failed gate blocks every dependent stage.

Recap: Conditional routing

"I was charged twice for order #4821."



LLM classifies the issue as **duplicate_charge** with 97% confidence.

Code maps **duplicate_charge** to the predefined **Billing Dispute** branch.

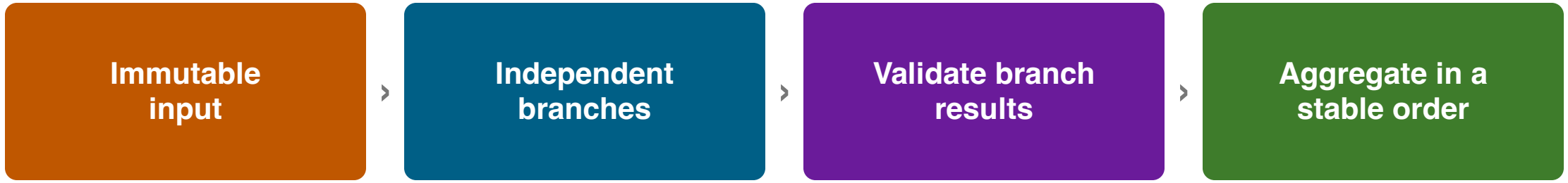
The billing branch retrieves the payment records and checks for duplicate charges.

The branch returns **duplicate_confirmed: true** to the shared **Draft Response** stage.

Use code for clear business rules. Use a model classifier only when the category requires semantic judgment.

Recap: Parallel fan-out and fan-in

"Order #4821 has two $49 charges. The customer requests a refund."



Every branch receives the same unchanged input

Run three checks simultaneously:

Payment check branch: Was the customer charged twice?
Policy check branch : Does the refund policy cover duplicate charges?
Fraud check branch: Does the account show suspicious activity?

Confirm that every branch returned the required data

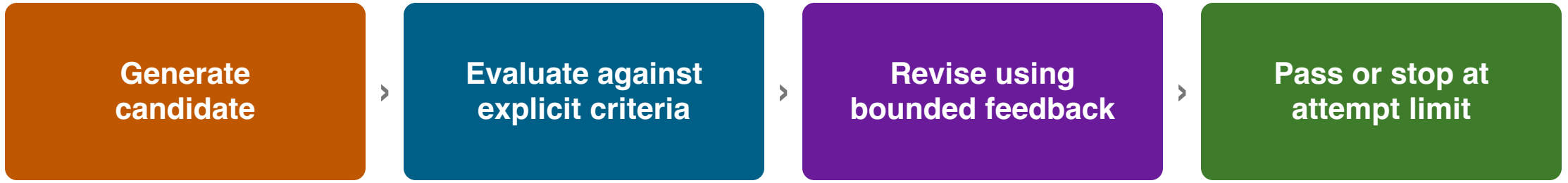
Combine results as payment → policy → fraud, then produce: Duplicate confirmed | Refund allowed | Fraud risk low

Recap: Three validation layers

- **Deterministic checks** cover reproducible, observable properties
- **LLM Judge:** Model evaluators score semantic quality against a written rubric
- **Human reviewers** handle consequential or subjective judgments
- A model critic can inform a decision but cannot serve as an authorization boundary

Recap: Evaluator and optimizer

e.g. “Improving a Refund Response”



LLM-> “Your $49 refund has been approved.”

The evaluator checks: correct amount, supporting evidence, required processing-time notice, and no unsupported promises.

Result: Fail. The response omits the processing time.

LLM-> “Your $49 refund has been approved. It should appear on your original payment method within 5–7 business days.”

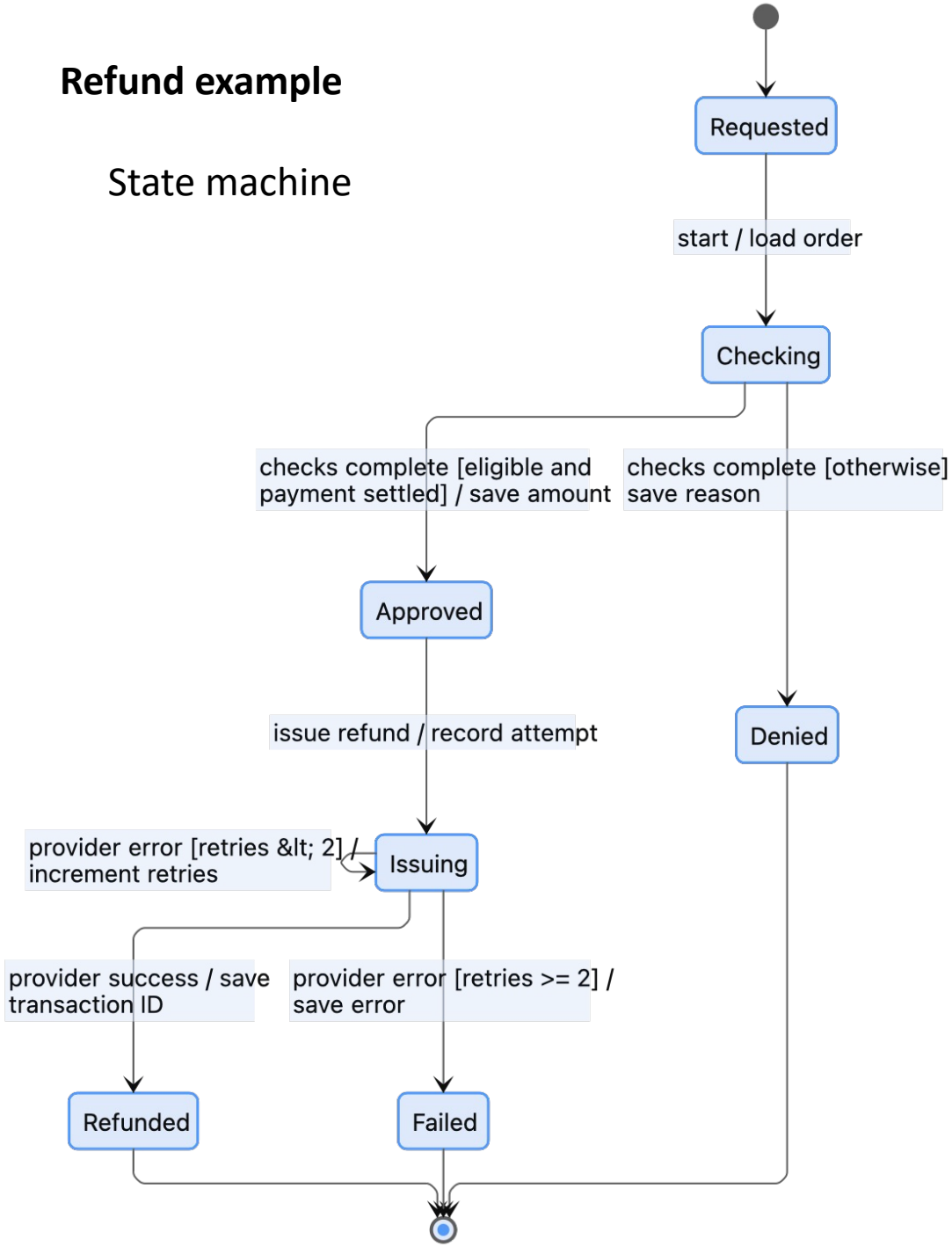
The revised response passes all criteria and moves forward

Recap: State machines and task graphs

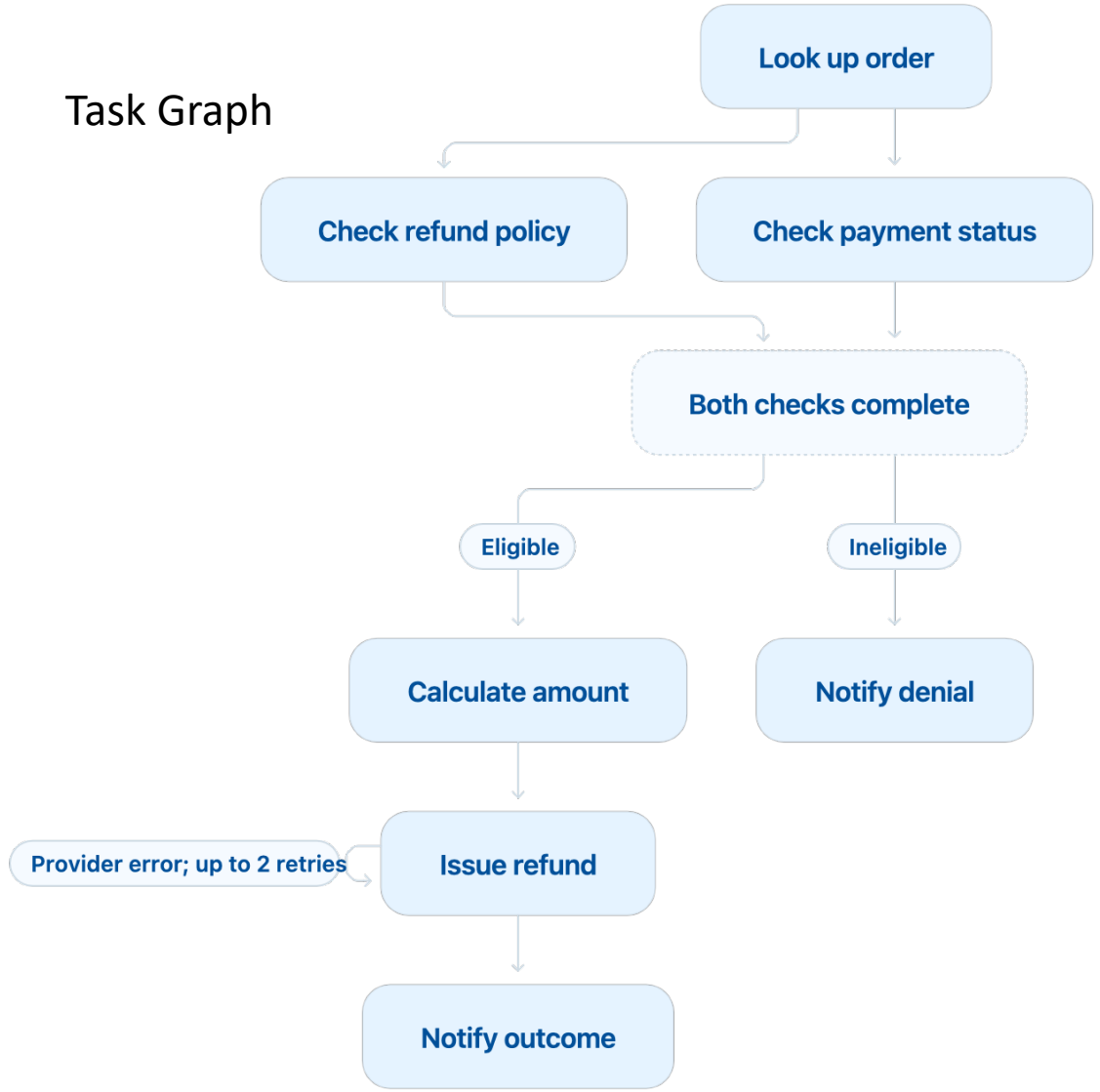
- A state machine defines lifecycle states, events, guards, updates, and terminal states
- A task graph shows execution dependencies, branches, joins, and bounded cycles
- One implementation may use both views
- State remains the source of truth
- Reject every illegal transition

Refund example

State machine



Task Graph



Recap: Workflow states

- **Typical progress states:** received, classified, analyzed, drafted, and validated
- **Terminal success state:** completed
- **Waiting state:** clarification_needed
- **Terminal exception states:** blocked or failed
- Each transition requires a named event and a satisfied guard

Week 5: Single Agent Loops and Adaptive Control

Week 5: Setup

- **Week 4:** Python selected each permitted transition
- **Week 5:** the model may choose the next approved action after observing a result
  - Python still owns tools, validation, authorization, budgets, and stopping

The changed question

- Not ***“does the system call tools?”***
- Ask ***“who determines the next action after each observation?”***

Adaptive selection is useful only when the next step cannot be fixed cheaply in advance

Exercise:

A user uploads a sales related CSV and asks for a short data story with appropriate visualizations. You haven't seen the file.

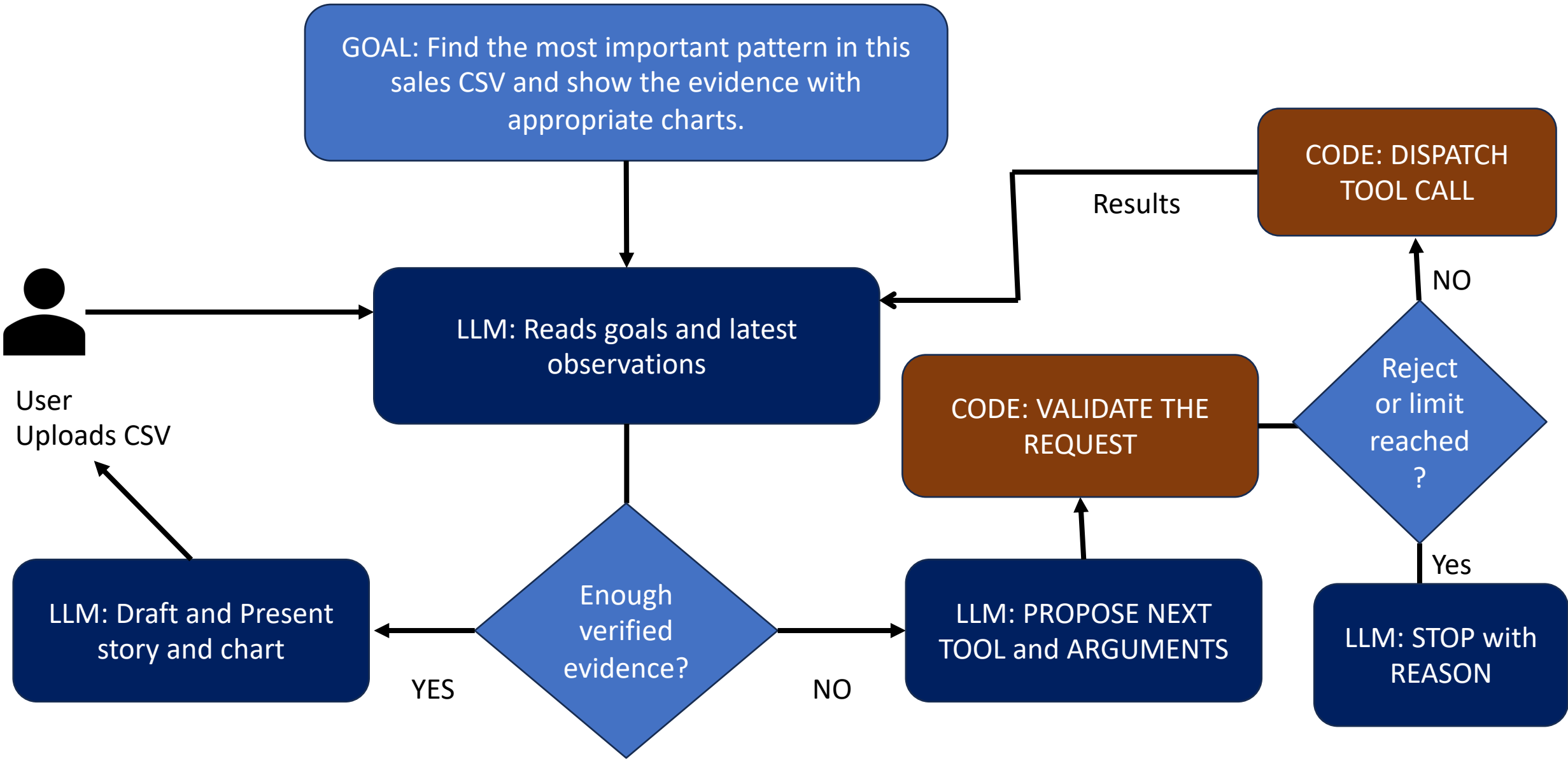
You build a deterministic workflow:

1. Read the CSV
2. classify columns as dates, numbers, or categories-
3. Apply chart rules: dates → line chart; categories → bar chart; numbers → histogram…
4. Fill a report template with summaries such as "Sales rose 12%."

What challenges might you face?

Challenges

- **Unknown columns:** Names and types do not always reveal what the data means.
- **Generic charts:** A valid chart may miss the question the data can answer.
- **Unexpected findings:** A spike or subgroup difference may need a new analysis.
- **Misleading patterns:** Missing values and outliers may distort the result.
- **Adaptive requirement:** The model must choose the next action after the data read; the application validates it and enforces limits.



Plan-react-observe LOOP

Suppose the user uploads this CSV

order_id	date	region	units	unit_price	status
N01	Nov 5	North	10	$20	completed
N02	Nov 20	South	5	$20	completed
D01	Dec 4	North	10	$20	completed
D02	Dec 15	South	5	$20	completed
D03	Dec 22	North	100	$20	completed
D04	Dec 27	South	50	$20	cancelled

Each step is a pass through the loop

- **Initial observations : EMPTY**
- **Inspect the CSV.** Tool call made and reports the columns and sample rows. The model sees that there is no revenue column and that cancelled orders must be handled.
- **Choose a calculation.** Model requests units × unit_price for *completed* orders, grouped by month.
  - The tool reports **$300 in November** and **$2,300 in December**.
- **React to the result.** That jump was not known at the start. The model requests the December order breakdown. T
  - The tool reports that order D03 contributes **$2,000 of December's $2,300**; D04 was cancelled and excluded.

Each step is a pass through the loop (…)

- **Choose and verify charts.** The model requests a bar chart comparing monthly completed revenue
  - a chart showing December's contribution by order. It checks that both charts total $2,300 for December.
- **Present the story.** "Completed revenue rose from $300 to $2,300, but one $2,000 order accounts for about 87% of December revenue. Without that order, December was $300—equal to November."

A minimal agent loop

- Receive goal
- Ask model for next action or answer
- Validate requested action
- Execute approved tool
- Return observation
- Update state
- Continue or terminate

Plan–act–observe–decide (ReAct)

- **Plan:** identify an appropriate next step
- **Act:** request an approved capability
- **Observe:** receive result or error
- **Decide:** continue, answer, clarify, block, or fail

ReAct in context

- Interleaves reasoning and actions
- Environment observations influence later decisions
- Originally evaluated on knowledge and interactive tasks

THE AGENT ENVIRONMENT

SINGLE AGENT

What the agent can observe, request, and change

01

AVAILABLE DATA

Files, records, and sources

02

AVAILABLE TOOLS

Operations the agent can request

03

OBSERVABLE STATE

Results, progress, and limits

04

ACTIONS

What a tool can read or change

05

PERMISSIONS

What is allowed and bounded

06

FAILURE BEHAVIOR

Errors, retries, and stopping

The CSV Example

Available data

Uploaded CSV and tool observations

Available read tools

inspect_csv, calculate, group_rows, inspect_rows, create_chart

State the application keeps

Goal, observations, completed calls, errors, budget, terminal reason

What the tools may do

Read the local file and make charts. The lab tools cannot modify external systems.

Capability comes from the exposed data and tools, under enforced permissions and failure rules.

In class activity (groups of 4)

Design a **university course-advising agent**.

Scenario: A student wants a viable next-semester schedule. Degree requirements, completed courses, preferences, and section availability may affect the recommendation.

- **Why adaptive agents needed:** Name two observations that could change the agent's next action. What does the model propose, and what does the application control?
- **Draw the loop:** Show **plan** → **act** → **observe** → **decide**, including the return to planning and possible endings.
- **Define the environment:** List available data, tools, observable state, actions, permissions, and failure behavior.

- **Time limit: strictly** 20 minutes

CAPABILITY VS AUTHORITY

- **Model capability:** can propose a useful action
- **Tool capability:** can perform an operation
- **User authority:** may permit a particular operation
- **Application policy:** decides whether execution is allowed
- These concepts must not be collapsed

Tool execution

- A tool description and schema tell the model how to request an action.
- The application checks the tool name, arguments, permissions, and limits.
- Approved requests run through fixed implementations that return structured results or errors.
- **A valid argument format does not imply authorization.**

State and bounds

- Store the goal, observations, completed calls, errors, and terminal reason explicitly.
- Enforce separate limits for turns, tool calls, retries, repetition, time, and cost.
- Stop with a stated outcome: completed, clarification needed, blocked, budget exhausted, or failed.

Plan first VS react first

- **Reactive loops:** Choose one action after each observation.
  - **Example:** Inspects the columns.
  - Seeing units and **unit_price**, it calculates revenue.
  - Seeing a December spike, it inspects December's orders. It chooses one step at a time.
- **Plan-first:** Outline steps before acting, then revise if evidence changes.
  - **Example:** Starts with a concrete plan: inspect columns, calculate monthly revenue, make a chart.
  - When the monthly totals reveal a spike, it revises the plan to inspect the underlying orders before charting.
- **Hybrid:** Keep broad milestones while selecting individual actions as results arrive.

Clarification and recovery

- **Ask when missing information** would change the action and cannot be safely discovered.
- **Treat tool errors as observations**: repair, retry, choose another approved action, or stop according to the error.
- **Do not repeat a failing request** without a reason.
- **Tool results are data**; they cannot change permissions or policy.

Completion and evaluation

- A final message alone does not establish that the goal was met.
- Check required evidence, unresolved requirements, and actions actually completed.
- Inspect the trajectory: requests, validated arguments, observations, state changes, limits, and terminal decision.
- Evaluate both the outcome and the path taken to reach it.

Over to Tutorial